# Activity 7: Front-end Web Development II

|  |  |
| --- | --- |
| Group # |  |

Participating group members

|  |  |
| --- | --- |
| Student ID | Name |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Part 1: Fetching Data from the PokéAPI Using JavaScript

## Objective

In this part, you will:

* Get hands-on with asynchronous JavaScript by experiencing Promises.
* Write code to fetch data from the PokéAPI.
* Complete a provided HTML/JavaScript template by filling in missing code.
* Explore key modern JavaScript features (such as Promises, arrow functions, and JSON parsing).

## Task 1.1: Experiencing a Promise

Before you jump into the PokéAPI Tester code, let’s see how JavaScript handles asynchronous actions using Promises.

1. **Open your browser’s console** (F12 or right-click → Inspect → Console).
2. **Type or paste the following code snippet:**

console.log("1. Starting fetch...");

fetch('https://pokeapi.co/api/v2/pokemon/1')

.then(response => response.json())

.then(data => console.log("3. Pokémon Data Loaded!", data));

console.log("2. This line runs BEFORE the data is loaded!");

1. **Observe the order of the messages:**

* Notice that "2. This line runs BEFORE the data is loaded!" appears before the Pokémon data is logged.
* **Question:** Why does the code behave this way?  
  *(Hint: Consider what a Promise represents and how asynchronous operations work in JavaScript.)*

1. **Write a short answer** (a few sentences) explaining your observation and your understanding of Promises.

The fetch instruction is synchronous that waits for the API request is complete.

|  |
| --- |
| **Hints: Understanding fetch() and Promises (Quick Overview)**  When we call fetch(url), it **doesn’t return data immediately**. Instead, it gives us a **Promise**—an object that represents a future result.  Think of it like **ordering food at a restaurant**:   * You **place an order** (call fetch()). * The **restaurant starts preparing the food** (the browser makes the API request). * You don’t get the food immediately! **You wait.** * When the food is ready, **the waiter brings it to you** (the .then() method handles the response).   In JavaScript, .then(response => response.json()) means:   1. **Wait for the response** to come back. 2. **Process the response** as JSON when it’s ready. |

## Task 1.2: Completing the PokéAPI Tester Code

Now, you will work with a simple HTML file that lets you enter a PokéAPI endpoint and query parameters. Some parts of the code are left incomplete. Your job is to fill in the missing pieces (marked with **// TODO:**).

**Starter Code**

Create a new HTML file and copy the code below. Then, complete the **TODO** sections.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>PokéAPI Tester - Part 1</title>

<style>

body { font-family: Arial, sans-serif; padding: 20px; background: #f9f9f9; }

input, button { padding: 8px; margin: 5px 0; width: 300px; }

pre { background: #efefef; padding: 10px; border-radius: 4px; overflow-x: auto; }

.container { max-width: 800px; }

</style>

</head>

<body>

<div class="container">

<h1>PokéAPI Tester - Part 1</h1>

<p>Enter the endpoint (after the base URL) and query parameters (if any).</p>

<p>Base URL: <code>https://pokeapi.co/api/v2/</code></p>

<form id="apiForm">

<label for="endpoint">Endpoint (e.g., <code>pokemon/1</code>):</label>

<input type="text" id="endpoint" name="endpoint" placeholder="pokemon/1" required>

<label for="params">Query Parameters (optional, e.g., <code>limit=10&amp;offset=0</code>):</label>

<input type="text" id="params" name="params" placeholder="limit=10&offset=0">

<button type="submit">Fetch Data</button>

</form>

<h2>API Response:</h2>

<pre id="response">Your formatted JSON response will appear here...</pre>

</div>

<script>

document.getElementById('apiForm').addEventListener('submit', function(e) {

e.preventDefault();

// Base URL for PokéAPI

const baseUrl = 'https://pokeapi.co/api/v2/';

// Get the user inputs

const endpoint = document.getElementById('endpoint').value.trim();

const params = document.getElementById('params').value.trim();

// Construct the full URL

let url = baseUrl + endpoint;

// If query parameters are provided, append them properly to the URL.

if (params !== "") {

url += '?' + params;

}

// Display a loading message

document.getElementById('response').textContent = 'Loading...';

// TO DO: Call fetch() to retrieve data from the constructed URL

// and show the data in <pre id="response">

});

</script>

</body>

</html>

You must use JSON.stringify(value, replacer, space) (<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify>) to format the retrieved data so that it is easy to read. Here is an example of what should appear on the UI.

A screenshot of a computer

Description automatically generated

Test your code using the following test cases.

**Test case 1**

A white rectangular object with black lines

Description automatically generated

**Copy the text result of the response from the API and paste it below.**

**Test case 2**

A white rectangular object with black lines

Description automatically generated

**Copy the text result of the response from the API and paste it below.**

## Task 1.3: Part 1 checkpoint

* Call a TA to demo your program.
* Submit this worksheet in myCourseVille